IoT System Project
Mini Project 1[E2]
(Smart Cabinet)

Group 10 Members

Armando Mak [223175Z]

R Sajeev [221325C]

1. Set up

. Power On and Connect ESP32 Devices

- **ESP32 Smartwatch**: Power on the device and ensure it's connected to Wi-Fi.
- **ESP32 CAM**: Power on the camera module and verify the camera feed is accessible.
- **ESP32 in Cabinet**: Ensure the device is powered and connected to the network.
- **Router:** Ensure the router is connected to the pwer

2. Set Up and Start the Flask Web Server

- **Navigate to Your Project Directory**:
  bash
  Copy code
  cd path_to_your_project_directory
- **Activate Virtual Environment (if used)**:
  bash
  Copy code
  source venv/bin/activate # Linux/Mac
- .\venv\Scripts\activate   # Windows
- **Start the Flask Server**:
  bash
  Copy code
  python app.py
- **Check Server Availability**: Open your browser and go to http://localhost:5000 to see if your server is running correctly.

3. Connect ESP32 Devices to the Server

- **Verify Wi-Fi Connection**: Ensure all your ESP32 devices are connected to the same Wi-Fi network as the server. Use Serial.print(WiFi.localIP()); on ESP32 devices to check their IP addresses.
- **Update ESP32 Code (if needed)**: Ensure that the server IP in your ESP32 code points to the correct IP address where the Flask

server is hosted.
cpp
Copy code
http.begin("http://IP-address"); // Replace with your actual server IP

4. Test Communication Between ESP32 Devices and Web Server

- **Send Data from ESP32 Devices**: Trigger actions on your ESP32 devices that send data to the Flask server (e.g., pressing a button on the cabinet).
- **Monitor Server Logs**: Check your server's console to verify that data is being received from the ESP32 devices.
- **Access the Web Interface**: Open the web interface in your browser to verify that data is being displayed correctly and that the controls work.

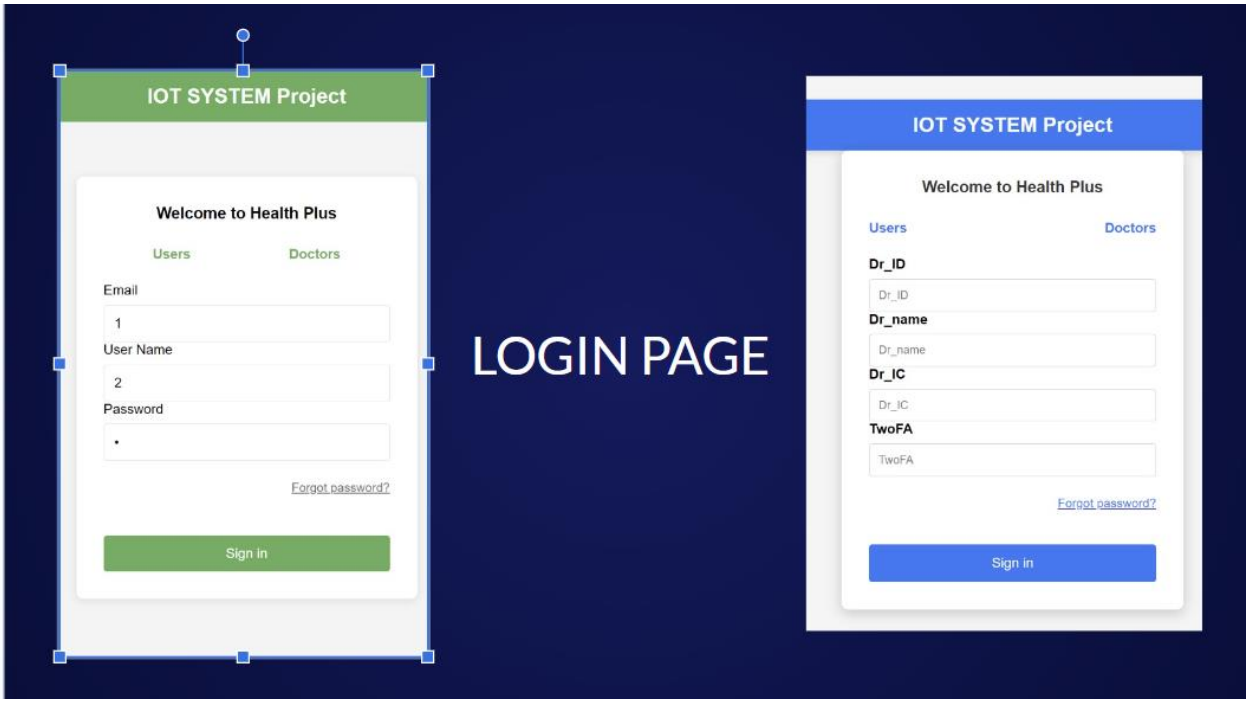
5. Network Configuration

- **Static IP for ESP32 Devices**: Assign static IP addresses to your ESP32 devices in your router settings to avoid reconfiguring IPs after a restart.

6. Secure Your Setup

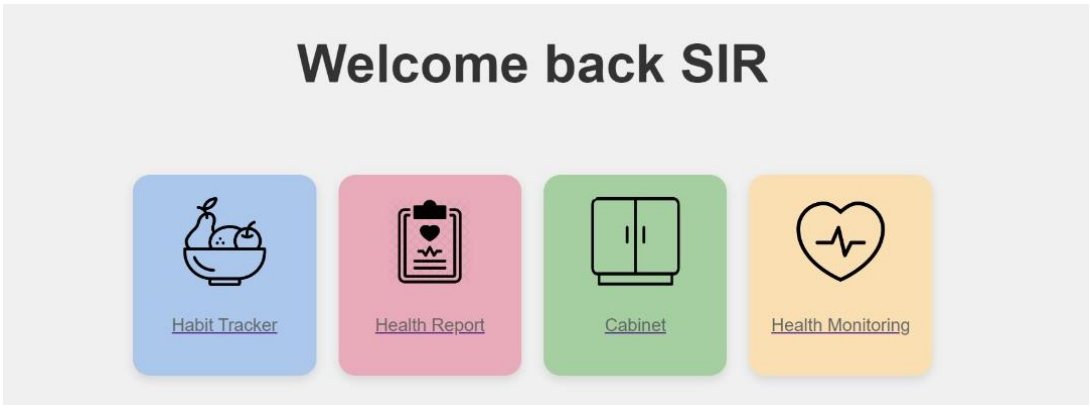
- **Use HTTPS**: Implement HTTPS on your Flask server using SSL certificates (e.g., Let's Encrypt) for secure communication.
- **Authentication**: Implement basic authentication for your web interface to prevent unauthorized access.

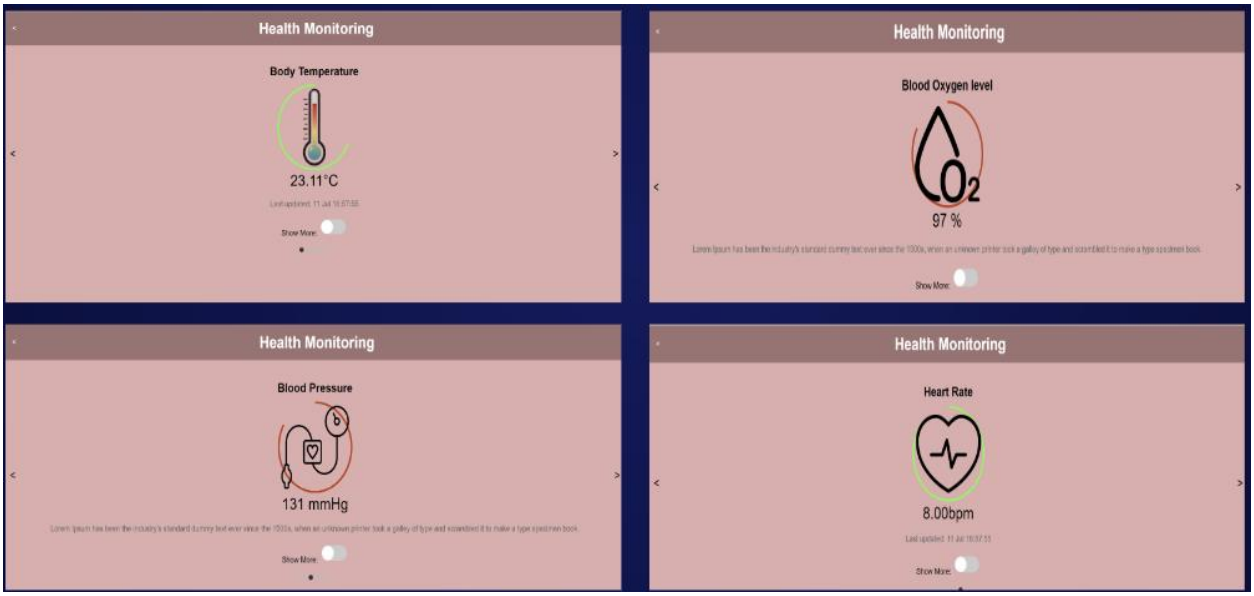
2. Dashboard graphics

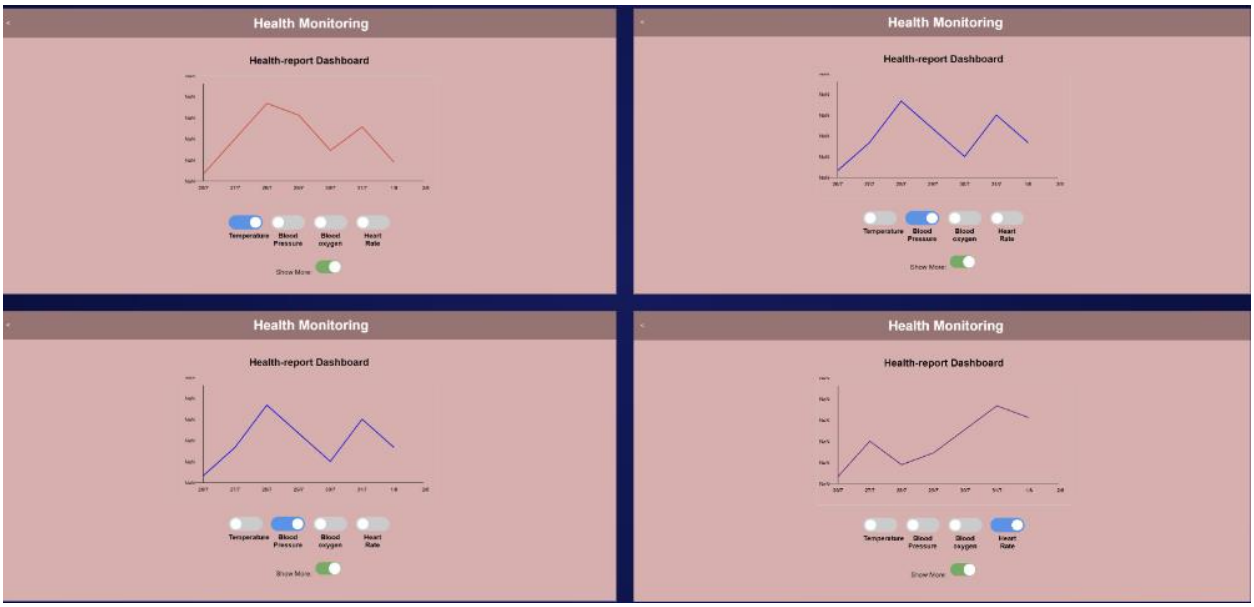


Data Sheet

Time	Heart Rate (HR)	Temperature (Temp)
11 Jul 08:49:58	44.00	23.11
11 Jul 08:50:28	44.00	23.11
11 Jul 08:51:03	28.00	23.11
11 Jul 08:51:33	52.00	23.11
11 Jul 08:52:08	48.00	23.11
11 Jul 08:52:38	48.00	23.11
11 Jul 08:53:13	48.00	23.11
11 Jul 08:53:48	56.00	23.11
11 Jul 09:04:11	88.00	23.11
11 Jul 09:04:41	60.00	23.11
11 Jul 09:05:11	152.00	23.11
11 Jul 11:21:01	8.00	23.11


Welcome back SIR
Habit Tracker
Health Report
Cabinet
Health Monitoring


Health Monitoring
Body Temperature
23.11°C
Health Monitoring
Blood Oxygen level
97 %
Health Monitoring
Blood Pressure
131 mmHg
Health Monitoring
Heart Rate
8.00bpm


Health Monitoring
Health-report Dashboard
Temperature
Blood Pressure
Blood oxygen
Heart Rate
Show More
Health Monitoring
Health-report Dashboard
Health Monitoring
Health-report Dashboard
Health Monitoring
Health-report Dashboard

Cabinet Items

Cabinet Items:

Nutella: 0

Pepsi: 1

Pringles: 0

Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

Health Report

view Report in PDF

Share with you doctor

Doctor's Note

Health Report

Blood Test Results

Total Cholesterol	1
Triglyceride	2
HDL-C	1
LDL	2
Chol/HDL	1
Bilirubin	2
ALT	1
AST	2
ALP	1
Golbulin	2
Protein	1
Albumin	21
Creatinine	1
Potassium	2
eGFR	3
HbA1c	4

Doctor's Advice

Cardiovascular: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Glucose: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Diet: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Exercise: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Mental: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Respiratory: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Digestive: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Bone Health: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Immunizations: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Download as PDF

Share with your doctor

Medications:

This includes all prescription and over-the-counter medications that the patient is taking, including vitamins and herbal supplements....

Immunizations

This includes a record of all the vaccinations the patient has received...

Social history

This includes information about the patient's living situation, marital status, occupation, and stress levels...

Sleep

How many hours of sleep do you typically get per night? Do you have trouble falling asleep or staying asleep?

Exercise

How often do you exercise? What type of exercise do you do? For how long?

Stress levels

Do you feel stressed on a daily basis? How do you manage stress?

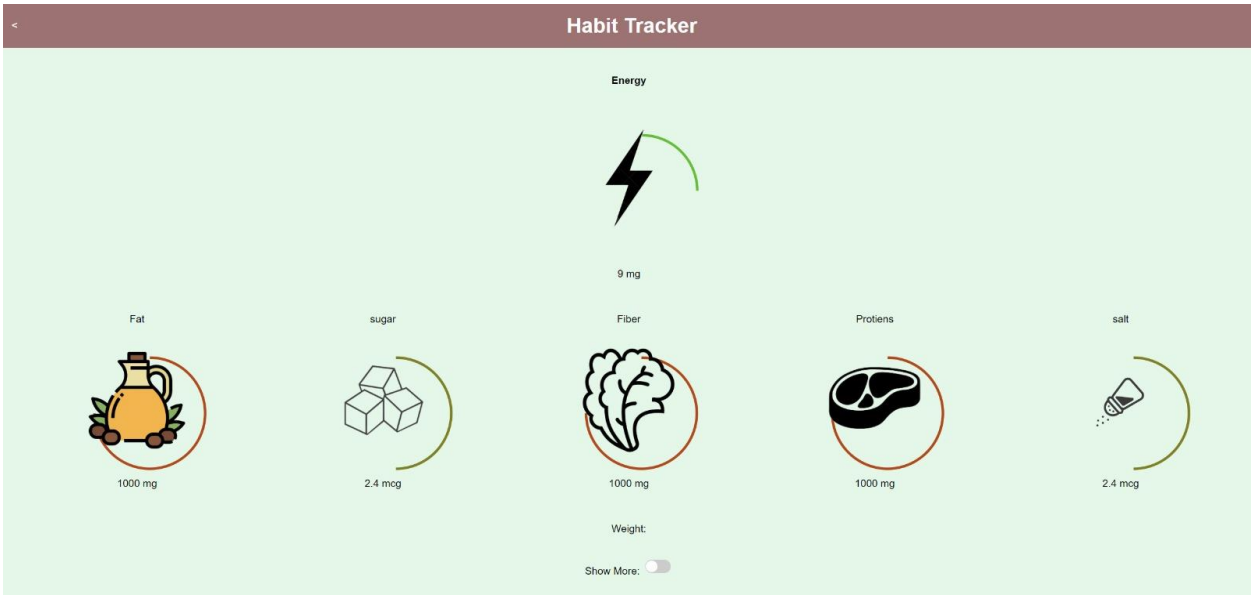
Alcohol and tobacco use

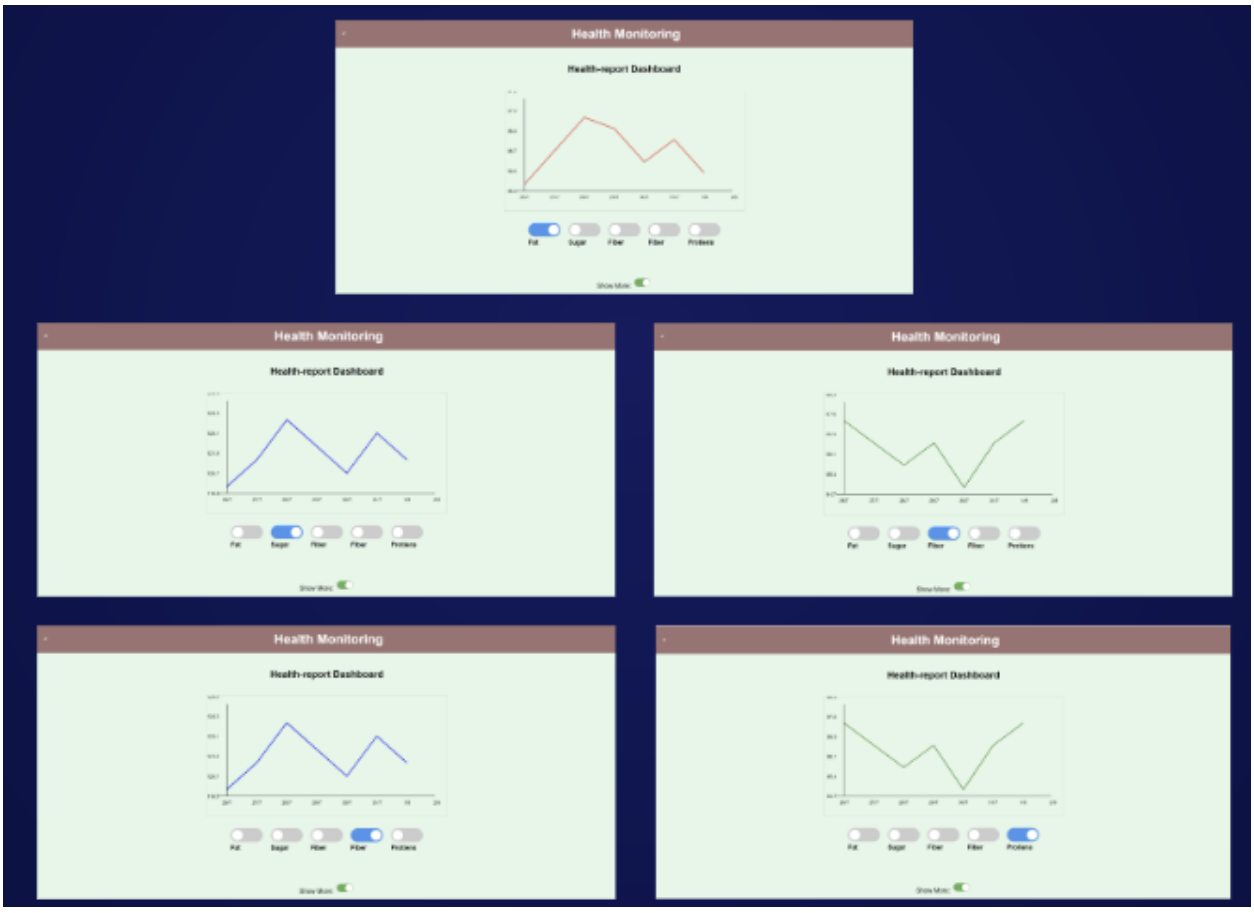
Do you drink alcohol? If so, how much and how often? Do you smoke?

Screen time

How much time do you spend looking at screens each day (TV, computer, phone)?

Back | Submit Notes


Habit Tracker
Energy
9 mg
Fat
1000 mg
sugar
2.4 mcg
Fiber
1000 mg
Protiens
1000 mg
salt
2.4 mcg
Weight:
Show More:


Health Monitoring
Health-report Dashboard
Health Monitoring
Health-report Dashboard
Health Monitoring
Health-report Dashboard
Health Monitoring
Health-report Dashboard

3. Setup for of sockets

Register New ESP client

When the esp client connects to the server

ESP-Client

```
def _init(SERVER_IP , SERVER_PORT , name):
    '''
        - For all devices connected to the server must follow this convention when connecting:
            - wait for server to send "id" to ur device
            - after received "id" send "new~(enter device id here)"
                - Convention is that device id is in 3 letters
            - wait for "send" from the server than u can send information.

        - This is to ensure that the server has registered the device and the device acknowledges this
        - all socket devices in this case send and receive info
    '''
    client_socket = socket.socket()
    client_socket.connect((SERVER_IP, SERVER_PORT)) # cinnect to server
    print("done power socket")

    while True:
        msg = client_socket.recv(35)
        print(msg)
        if b"id" in msg: #wait for the request for id declaration
            break
```

- The ESP client on power up will try to initialize the socket object and connect to the server
    - ```
      client_socket = socket.socket()
      client_socket.connect((SERVER_IP, SERVER_PORT)) # cinnect to server
      ```
- Then it will wait for the server to request for the client for its id
    - ```
      msg = client_socket.recv(35)
      ```

|

V

Server

```
# Define the socket_connector function
def socket_connector():
    while True:
        clientsocket, address = s.accept()
        if clientsocket.getpeername() not in client_to_send.keys():
            client_to_send[clientsocket.getpeername()] = [clientsocket] # init of keys
            print(client_to_send)
            print(f"Connection from {address}, {clientsocket}")
            connection_estab = "id pls"
            connection_estab = f"{len(connection_estab):<{HEADER_SIZE}}" + connection_estab
            clientsocket.send(bytes(connection_estab, "utf-8"))

        listener = threading.Thread(target=socket_listener, args=(clientsocket,))
        listener.start()
```

- Run the above code in a thread

- In this code it will accept any socket that tries to connect to itself
  - ```
    clientsocket, address = s.accept()
    ```
- Then check in the client to send if the socket was registered before
  - If it does not it will request id from the ESP client
  - ```
    connection_estab = "id pls"
    connection_estab = f"{len(connection_estab):<{HEADER_SIZE}}" + connection_es
    clientsocket.send(bytes(connection_estab, "utf-8"))
    ```
- Then start the message listener “socket_listener”
  - ```
    listener = threading.Thread(target=socket_listener, args=(clientsocket,))
    listener.start()
    ```

|
V

Initialize relevant info

ESP-Client

- Client then sends “new~”
  - ```
    '''
    client_socket.send(id_name)
    ```
- Which is to initialize all the other relevant info in client_to_send
- Then wait for server to send “start”
  - ```
    while b"start" not in client_socket.recv(32): # acts as a hang sta
        continue
    ```

|
V

ESP Server

- Receives the “new~”
  - ```
    data = clientsocket.recv(fb_size)
    ```
- Checks for new client
  - ```
    if check_new_client(client_to_send ,remote_addr) == False:
    ```
- If it is a new client:
  - Check if the remote address is inside the client_to_send dictionary and has the “new~” message
    - ```
      if remote_addr == keys and b"new~" in data:
      ```
  - Put the necessary information
    - ```
      client_to_send[remote_addr].extend([id , True , STD_FB])
      ```
    - id
      - for convenience on coding side as we can just call “cab” instead of its remote address to find its information
    - True
      - (unused)
    - STD_FB
      ```

- To set the standard frame buffer size = 2048 bytes

```
STD_FB = 2048
```

- Send the “start”
  -
```
client_to_send[remote_addr][0].send(b"start")
```

|

V

ESP-Client

- Then allow the rest of the code to proceed
  -
```
return client_socket
```

---

4 & 5. Sending info using sockets

- sending can only be in utf-8 or base64
- The convention to send information from the esp client looks like: https://www.canva.com/design/DAGJTyrvH_w/cjLpcchwf4Gd9K_r6uydag/edit?utm_content=DAGJTyrvH_w&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton
- Convention 1 is to ensure there is always space for data packets to come into the server

```
client_socket.send(b"size-{}".format(len(msg))) # sends size declaration
while b"send" not in client_socket.recv(32): # waits for "send" message by server. acts as a hang statement
    continue
client_socket.send(msg) # sends message
return True
```

- Convention 2 just sends and does not declare the size:

```
    while True:
        try:
            data = s.recv(1024)  # <8 = ok
            if data == b"bpm":
                bpm_value = Hr_sensor.calculate_bpm_over_15sec()
                temperature = temp_sensor.read_object_temp()  # Correct way to read temperature
                s.send(f"{bpm_value:.2f},bpm".encode("utf-8"))
                s.send(f"{temperature:.2f},temp".encode("utf-8"))
        except OSError as e:
            print("Socket error:", e)
            # Handle the error as needed, e.g., reconnect or continue
```

- Just to add on, all the esp32 can do threading except the ESP32 CAM. As it will mess with its firmware, which results in error messages.

Convention 1

This method only applies from ESP client to server

- This is to ensure no matter what the frame buffer will always be bigger
- Can be seen in the camera and the cabinet

Steps:

For the ESP client:

1. For the esp-client it first has to send the size declaration
   - `client_socket.send(b"size-{}".format(len(msg))) # sends size declaration`
   - Then it waits for the send
     -
```
while b"send" not in client_socket.recv(32): # waits for "send" message by se
    continue
```

On the server side:

2. It will first search for the highest Frame buffer size from all the known users(default is 2048 bytes `STD_FB = 2048`)
3. Then it will receive any information at a frame buffer of the highest known frame buffer from all its users
   - `fb_size = largest_fb(client_to_send)`
4. Receive the data at the highest frame buffer size
   - `data = clientsocket.recv(fb_size) #`
5. Then find the id from the remote address
   - `id = find_clients_id(client_to_send ,remote_addr)`
6. See if it is a new client
   - `if check_new_client(client_to_send ,remote_addr) == False:`
7. If it is not a new client, check the message and see if it is a size declaration or a message:
   a. Size Declaration
      - If it is a size declaration:`if b"size-" in data: # size declaration`
        - Only take the size of the frame buffer and assign it to the esp-client's frame buffer size
        - `client_to_send[remote_addr][3] = int(data[5:].decode("utf-8"))`
        - As typically the send is sent with "size-FrameBufferValue". Where FrameBufferValue is the size of the message to be sent
        - Then send the acknowledgement "send"
        - `client_to_send[remote_addr][0].send(b"send")`

      For the ESP client:
        - When the ESP client receives the "send" acknowledgement, then it can send the information
          -
```
client_socket.send(msg) # sends message
return True
```

   b. Message:
      - Repeat steps 2 to 4
      - Find the id of the ESP client sender
        - `elif b"img" == id:`

Convention 2

It can be done for

- Server to ESP client
    - The messages sent from the server to the clients are typically 3-4 letters/characters long
- ESP client to Server
- The frame buffer size will always stay the same no matter what
- Can be seen in the camera and the cabinet
    - Partly seen in smart watch

Server to ESP client

Steps:

Server

- Server first has to find the id to send
    -
```
for keys in client_to_send.keys():
    print(f"stuff {client_to_send[keys][1]}")
    if b"img" == client_to_send[keys][1]:
```
- Send the message
    -
```
client_to_send[keys][0].send(b"img")  # Send image request to client
```

ESP client:

- Waits for the message to be received
    -
```
req_server = client_socket.recv(35)
```
    - In this case the frame buffer is 35
        - I chose 35 bytes because it should be more than enough to accommodate 3-4 letter messages
- Use the message
    - In this case if the message is lof

```
if req_server == b"lof": # request to close lights
```

        - close all the lights

```
led_rgb_setter(led_strip  , NO_OF_LED , 0 , 0 ,0)
```

Server to ESP client to Server

- This is only seen in the smartwatch
    - As the server will keep requesting for the smart watch every 30 seconds
    - This is so to ensure the server will not be overloaded with all the messages
        - As it will be ready to accept the smart watch
- It is similar to **Server to ESP client**

Steps:

Server

- It is first initialized by a scheduler

- ```
  scheduler.add_job(send_data_request_handler, 'interval', seconds=5)  # Adjust timing here
  ```
- First check if 30s has elapsed since the last sent time or initialized time
  - ```
    if last_data_time and (datetime.now() - last_data_time).total_seconds() < 30:
        return "Waiting for 30 seconds after the last data reception."
    ```
- **Server to ESP client (same process)**
  - Code used:

```
# Define the send_data_request function
def send_data_request(data_type):
    global last_data_time

    if not client_to_send:
        return "No client connected"  # Return if no client connected

    for client_addr, client_data in client_to_send.items():
        if len(client_data) >= 2 and b"wch" == client_data[1]:
            print(f"Sending {data_type} request to client {client_addr}")
            client_data[0].send(data_type.encode())  # Send data type request to client
            last_data_time = datetime.now()

    return "sent"  # Return sent
```

ESP Client

- Data is then sent
  - ```
    s.send(f"{bpm_value:.2f},bpm".encode("utf-8"))
    s.send(f"{temperature:.2f},temp".encode("utf-8"))
    ```

Server

- It will first search for the highest Frame buffer size from all the known users(default is 2048 bytes `STD_FB = 2048`)
  - Since we **never** declared the size, the highest will be whatever the highest frame buffer from other users in that time
  - However, no matter what, the smallest size will still be 2048 bytes because the size declaration never changed as we never initialized it
- Then it will receive any information at a frame buffer of the highest known frame buffer from all its users
  - ```
    fb_size = largest_fb(client_to_send)
    ```
- Receive the data at the highest frame buffer size
  - ```
    data = clientsocket.recv(fb_size) #
    ```
- Then find the id from the remote address
  - ```
    id = find_clients_id(client_to_send ,remote_addr)
    ```
- See if it is a new client
  - ```
    if check_new_client(client_to_send ,remote_addr) == False:
    ```
- If it is not a new client find the id of the ESP client sender
  - ```
    elif b"wch" == id:
    ```